A DATA BROKERING METHOD AND SYSTEM

Field of the invention

The invention relates to the field of the synchronisation of data and in particular to a data brokering method and system for the synchronisation of data in a geographically load-balanced environment.

Background of the invention

Distributed systems allow for the sharing of resources. The Internet is an example of a distributed system with its vast interconnected collection of computer networks of many different types. Other examples of networks include mobile phone networks, corporate intranets and home networks etc. Computers are connected to these networks to allow a computer to communicate with other computers. These computers may be spatially separated by any distance (for example in separate continents, in the same building or the same room).

When a computer is connected to a network, it is possible for a computer to share one or more resources for example, hardware components such as printers and disk drives and software components such as data files, databases and other data objects.

To allow the sharing of a resource, a computer program executing on a computer communicates with other computers by passing messages. An example of this is the HyperText Transfer Protocol (HTTP), which is a request reply protocol in which a client device sends a request message to the server containing the URL of the required resource. The server looks up the path name of the URL and if it exists, the server sends back the requested resource in a reply message to the client device.

In a distributed system it is likely that more than one client device will want to access a shared resource at the same time for example, a number of people using an on-line auction database. Using the example of the on-line auction database, if there are 3 multiple concurrent bids from 3 different people for example, Hall bidding £123, Robertson bidding £124 and Midgley bidding £135 and the bid data is interleaved, this could lead to the auction system storing the data as Hall £135, Robertson £123 and Midgley £124 and hence storing incorrect bids in relation to the people bidding.

Therefore in a distributed system a shared resource (the auction system) must be responsible for ensuring that it operates correctly in a concurrent environment. For a shared resource to operate correctly, its operations must be synchronised in such a way that its data remains consistent.

How to provide consistent data in real time is a problem faced by web service providers and suppliers. As the Internet has developed and the types of web services that are provided have evolved into more data intensive dynamic content, web service providers have to find a way to manage tremendous amounts of data in an efficient manner.

This problem is further compounded by web service providers wanting to implement solutions for a global market, but want to provide customers in every geographical region of the global market with a high availability of service and good performance.

Prior art solutions to the above problem have involved web service providers and suppliers adopting a distributed model in the implementation of a web service. A distributed model requires one or more web services being deployed across a number of geographically dispersed physical environments. At the same time each of these physical environments require a consistent view of some of the data that is common to all implemented web services. This creates a challenge and adds a layer of complexity to not only synchronising data amongst each of the implemented web services, but also managing the process of having multiple systems attempting to update the same data.

In order to update data across multiple servers prior art systems employ a protocol called 'two phase commit', which allows each of the servers involved in a transaction to either accept an update or to rollback an update, thereby maintaining consistent data. To achieve this, one of the servers takes on a coordinator role, which involves ensuring the same outcome on all the serves. Two phase commit allows a client device to send a request to commit or rollback a transaction to the coordinator server. If the request is to abort a transaction the coordinator informs all other participating servers immediately and the transaction is rolled back. If the request is to commit a transaction the coordinator sends a request to all the participants asking if they are prepared to commit the transaction. If a participant can commit the transaction it will commit as soon as the appropriate records have been updated in permanent storage and the participant is prepared to commit.

A disadvantage of using two phase commit is that the protocol is dependent on the implementation of specific products supporting the two phase commit protocol and restricts the implementation to the same technology at all physical environments, for example, it is not possible to deploy multiple vendor solutions across multiple physical environments. Therefore two phase commit does not provide a solution that is vendor neutral as two phase commit is implemented within the update/retrieval mechanism itself. This poses a significant problem when required to implement solutions that require geographical load balancing of web infrastructures to provide resilience in the event of a loss of one of the physical sites.

US patent application publication 2002/0188610, discloses a data storage and access mechanism employing clustering. The data management system comprises a plurality of application servers, web servers and data servers. The data management system also includes a session manager directing users accessing the system to a subset of web servers, application servers and data server's bases on the characteristics of the users.

US patent application publication 2002/0188610 discloses that there are two forms of replication strategies, master-slave and master-master mode and describes the disadvantages of both approaches. In particular the master-master replication strategy is disclosed as difficult to design and implement and US patent application publication 2002/0188610 further states that architects must create their own conflict resolution mechanisms as well as their own work delegation techniques. Further because the data exists in two different computer systems at the same time, there is nothing stopping two different users from changing the same piece of data. Finally US patent application publication 2002/0188610 states that a failure in the master-master approach would leave both systems out of sync and could cause data instabilities in other parts of the system until the problems are found and corrected.

Therefore there is a need to provide a method and a system for the synchronisation of data in a geographically load balanced environment and address the shortcomings of the prior art as discussed above.

Disclosure of the invention

Viewed from a first aspect the present invention provides a method for the synchronisation of data in a geographically load-balanced environment comprising a database broker, the database broker being operable as a

middleware service, the method comprising the steps of: receiving a request from an application layer to update a dataset owned by the database broker, the dataset being for use by an update mechanism; first determining if the dataset for use by the update mechanism belongs to a common database or a local database; second determining if the database broker receiving the request is a master database broker or a secondary database broker; and in response to a first and second successful determining step the owning database broker updating the dataset in the common database or the local database.

Preferably the present invention provides a database broker service for managing database updates across a plurality of web sites. The database broker is implemented as an application middleware service, interfacing to the application layer and application server which in turn requires access to the update/retrieval mechanism comprising the data.

The invention operates independently of the underlying update/retrieval mechanism and is therefore vendor neutral. This therefore enable customers to deploy global load balancing solutions with any database technology and removes the need to implement complex two-phase commit processing at the physical database level, and instead provides application level distribution and aggregation. The invention further enables scalability up to multiple sites, beyond simply dual site implementations.

In order for a database broker to have a 'global view' of the data each database broker is provided with an instance of each database. Although each database broker is able to view all of the data required to provide the web service, each database broker is given ownership of a dataset that only they can update. In order to update a dataset, the update is packaged and sent to the owning database broker of the dataset. The owning database broker will validate and coordinate the update, for example, when simultaneous updates are issued by one or more database brokers competing to update the same resource. Once the owner is satisfied and has processed the update itself, an acknowledgement will be returned to a requesting broker, and only then will an update be performed.

Advantageously, each dataset within the update/retrieval mechanism is divided into common data or local data. The common data comprises data that is critical to all servers for example, the amount of stock available. The common data needs to be replicated across all websites with read access by the database brokers. Preferably a master database broker is designated as the owner of the common database and it is only the master database broker that

can update the common dataset. Preferably at least one secondary database broker is designated as the owner of a dataset within a local database. A local database comprises data that is relevant to a geographical region for example details of the names and addresses of a plurality of U.S. customers. Additionally the local database may comprise data which is mastered locally by a database broker, and that does not form part of the common dataset. The local data will also need to be replicated across all services with read access by the database brokers.

Preferably, a database broker registers itself at server initialisation time as a master database broker or a secondary database broker depending if a master database broker has already been elected and flags its status as active.

Advantageously, the present invention provides for a monitoring component probing at least one of the database brokers to determine if the database broker is still active. It is possible for a database broker to become inactive due to failures in the system such as a server crash.

Preferably if it is found that a database broker is not active, the database broker is removed from a list storing the details about each active database broker registered in the system.

An advantage of the invention is that it is self optimising. If the monitoring component detects that a database broker is failing, a further check is performed to determine if it is the master database broker that is failing. If the response is positive a lookup if performed in a configuration database or other data store to determine whether there is a delegate database broker defined to 'take over' as the master database broker from the inactive master database broker. This provides an advantage in that any number of 'chains of commands' can be defined if a master database broker fails, ensuring the high availability of the web service. The chain of command may be extended to any number of secondary database brokers as well as a master database broker. It is further possible for a database broker to be a master database broker as well as a secondary broker.

Preferably a database broker maintains a log of the most performed queries held in its cache and uses the log to identify the most common queries performed. The database broker pre-loads this data into the cache when the broker registers in the system. The master broker aggregates the most common queries from all participating brokers, and propagates this information around

all database brokers so that the entire system may be pre-loaded with most common query data on a restart.

Viewed from another aspect the present invention provides a database broker system for the synchronization of data in a geographically load-balanced environment, the database broker system being operable as a middleware service, the system comprising: a query module comprising means for managing queries received from an application layer and sending and receiving information from an update mechanism; an update module comprising means for synchronizing at least one requested update between the at least one database broker; a configuration module comprising means for receiving data from an application layer and storing configuration data about the database broker and means for receiving updates from a monitoring component; and a monitoring module comprising means for determining if the database broker is active and means for sending one or more updates to the update module.

Preferably the system interfaces with an update/retrieval mechanism and is implemented as a middleware service. Preferably the present invention provides for the update/retrieval mechanism comprising means for the segmentation of at least one database into to logical datasets.

Advantageously, the present invention provides for the monitoring component comprising means for probing the at least one database broker and means for determining if the at least one database broker is active.

Preferably the present invention provides for the monitoring component further comprising means for determining if the master database broker is failing.

Preferably the present invention provides means locating a delegate database broker in response to a successful determining means.

Preferably the present invention provides means for locating a delegate database broker is determined by means for the configuration module defining one or more delegate relationships.

Viewed from another aspect the present invention provides a computer program product comprising computer program code stored on a computer readable storage medium, which when executed on a data processing system, instructs the data processing system to carry out the invention as described above.

Viewed from another aspect the present invention provides a data synchronisation service in a geographically load balanced environment comprising a database broker, the database broker being implemented as a middleware service, the service comprising the steps of: receiving a request from an application layer to update a dataset owned by the database broker, the dataset being for use by an update mechanism; first determining if the dataset for use by the update mechanism belongs to a common database or a local database; second determining if the database broker receiving the request is a master database broker or a secondary database broker; and in response to a first and second successful determining step the owning database broker updating the dataset in the common database or the local database.

Brief description of the drawings

Embodiments of the present invention will now be described, by way of examples only, with reference to the accompanying drawings in which:

Figure 1 is a block diagram depicting a multiple of websites in which a preferred embodiment of the present invention may be applied;

Figure 2 is a block diagram detailing the components of a database broker according to a preferred embodiment of the present invention;

Figure 3 is a block diagram detailing the update component of Figure 2 in accordance with a preferred embodiment of the present invention;

Figure 4 is a block diagram detailing the configuration component of Figure 2 in accordance with a preferred embodiment of the present invention;

Figure 5 is a block diagram detailing the query component of Figure 2 in accordance with a preferred embodiment of the present invention;

Figure 6 is a block diagram detailing the monitoring component of Figure 2 in accordance with a preferred embodiment of the present invention;

Figure 7 is a block diagram detailing the segmentation of data in the update/retrieval component of Figure 2 in accordance with a preferred embodiment of the present invention;

Figure 8 is a block diagram of a plurality of geographically dispersed web sites in accordance with a preferred embodiment of the present invention;

Figure 9 is a flow chart detailing how an update is performed on a dataset by a database broker in accordance with a preferred embodiment of the present invention;

Figure 10 is a flow chart detailing the interaction between multiple database brokers; and

Figure 11 is a block diagram detailing a one or more relationships that may be created between one or more database brokers.

Detailed description of the invention

When providing web services over multiple geographical locations, users require high availability services. This requires giving users access to a web service within a reasonable response time. At the same time all websites need to have a 'global view' of the data. For example, in a stock system there may be 50 particular door handles in stock. It is crucial that this information is replicated to all servers; otherwise, for example, a user of a website located in the UK may have different information on the availability of a door handle to a user of the same website located in the United States.

In order to achieve a 'global view' of the data in real time, a database broker implemented as a middleware service is provided which assumes responsibility for its own dataset within a database. A middleware service is a software layer which 'sits' above the operating system and whose purpose is to mask the heterogeneity of the operating system environment. Middleware is concerned with enabling the implementation of communication and resource sharing support for distributed systems. Implementing the database broker in middleware allows the database broker to be platform and vendor neutral.

Referring to Figure 1, two typical web sites 'xyz.com' 10, 12 located at two different geographical locations are illustrated. The web sites 'xyz.com' 10, 12 may provide one or more web services for example an on-line banking service or an on-line auction system. Each web site 10, 12 may comprise the following components: a web server 100 and 104 for serving web pages and web resources such as images, style sheets and data files etc to the website 'xyz.com' 10, 12. An application server 101 and 105 for managing applications that are required by the one or more web services for example, a chat

interface wherein a user of a client device can ask questions to a customer services advisor at their bank over a network 11 which is received and processed by the application server 100, 105. A database broker 102, 106 which is implemented as a middleware service for providing the management of database updates across multiple websites located at the same or different geographical locations and an update/retrieval mechanism 103, 107 for the storage, updating and retrieval of a plurality of data required by a web service.

All of the above components interact with each other to exchange messages in order to provide a service requested by one or more client devices (not shown).

Communication between a plurality of database brokers 102, 106 may be enabled over a private network 108. The private network 108 may comprise any suitable communication medium such as for example, Ethernet or Fiber Optic etc. The database brokers may exchange messages between one and other across the private network relating to the database updates to be performed. It will be obvious to a person skilled in the art that it is also possible for the database brokers 102, 106 to communicate with each other over other types of networks, for example, the Internet or an Intranet etc.

Referring to Figure 2, the components of the database broker are illustrated. The database broker comprises a request manager component 225 to manage one or more application requests, a query component 200 to perform caching of data requested from an update/retrieval mechanism 220, a configuration mechanism 205 for storing and retrieving a 'chain of command' should a database broker fail, an update component 210 for processing the requested updates within the system, a monitoring component 215 for determining whether a database broker 102, 104 is available for service and an update/retrieval mechanism 220 for the segmentation of data and storing and retrieving of updates.

Each of the above components will be further explained with reference to Figures 3 to 8.

Figure 3 discloses the update component 210 of the database broker 102 of Figure 2. The update component 210 receives one or more API requests from the request manager 225 of Figure 2 which processes the API requests sent from the application server and extracts one or more data objects. The data objects are routed either to the query component 200 or the update component 210 depending on the type of request received.

The update component 210 comprises an update manager 300 which provides the overall management of the updates by overseeing that an update is processed correctly between the update component 210 and the monitoring component 215.

All updates performed by the update manager 300 are recorded in an update log 305. The update log 305 comprises information on local updates performed by the receiving update manager and updates performed by other database brokers 102, 106 within the system. The information within the update log 305 is accessed by a synchronization manager 415 to compare the status of a local database broker 102 with the status of one or more remote database brokers 106 within the system.

The synchronization manager 315 manages the synchronization of one or more updates between one or more database brokers 102, 106 and responds to events received from the monitoring component 205 and ensures that requested updates are not interleaved with another to produce inconsistent data. Finally, a sequence manager 310 generates sequence numbers which are distributed and tagged to the requested updates to ensure that an update is performed in a correct sequence.

Referring to Figure 4, the components that comprise the configuration component 205 of Figure 2 are illustrated in further detail. The configuration component 205 comprises a configuration manager 405 and an administration interface 400. An administration client (not shown) interacts with the administration interface 400. Through this interface the administration client interacts with the configuration manager 405.

The administration interface 400 will reflect the configuration of the system, as managed by the configuration manager 405, based on the level of detail authorized by the administrator's credentials. Different levels of administration may be performed with differing levels of scope from a single database broker 102, to a localized domain of database brokers 102 or to all the database brokers 102, 106 that are configured across the entire system. Each database broker 102, 106 will contain the configuration details of the entire system as this information is replicated amongst the brokers to ensure a consistent view, as well as a distributed administration capability.

The configuration manager 405 interacts with the monitoring component 215 and provides the configuration settings for the database brokers 102, 106.

The configuration setting may comprise the name of the database broker 102, 106, the 'chain of command' should the database broker 102, 106 fail, the elapsed time since the last communication received and how long has the database broker 102, 106 been active for. The information gathered by the configuration manager 405 is accessed by the monitoring component 215 when probing one or more database brokers 102, 106 to determine if each registered database broker 102, 106 is active.

Now turning to Figure 5, the query component 200 of Figure 2 can be explained in greater detail. The query component 200 comprises the following components: a query manager 500, a query cache log 505, a query cache 515 and a query engine 510. The query manager 500 receives a plurality of request from a request management component (not shown) for example how many stainless steel door handles are there is stock. The query manager 500 sends and receives queries to the query cache log component 505. The query cache log 505 component maintains the query cache log 505 which is used to perform pre-fetching and loading of the cache, and cache pre-loading (based on previously saved most commonly performed queries). The query manager 500 sends and receives updates from the query cache 515 for example a cached query, i.e. how many door handles in stock.

The query manager 500 initially receives the query request details from the request management process 225 of Figure 2. The query manager 500 sends and receives updates to the query engine 510 which processes a query, including the aggregation of one or more parallel queries performed when there is a plurality of data stores within the system.

For example, an application issues the query to the database broker requesting the total number of door handles which are in stock. The request is received by the request manager 225 in Figure 2, and is passed to the query manager 500. The query manager 500 then checks the query cache 515 for a recent and matching query. If the query is matched, the data is retrieved and the results returned. If the query is not matched, either because the cache data has expired or no cached data could be matched, then the query manager 500 identifies the target database which holds the query dataset based using the information held by the resource manager 600 in Figure 6. In this instance the dataset is held in the common dataset, and the query manager 500 constructs the query request and passes it to the query engine 510 for processing.

The query engine 510 issues the query to the common dataset and retrieves the result. This is passed back to the query manager 500, which in turn updates the query cache 515 and the query cache log 505. The query manager 500 returns the result to the application server 101.

In another embodiment, a query is issued by the application for data contained within the local dataset. The request is received by the request manager 225 in Figure 2, and is passed to the query manager 500. The query manager 500 checks the query cache 515 for a recent and matching query. If the query is matched, the data is retrieved and the results returned. If the query is not matched, either because the cache data has expired or no cached data could be matched, then the query manager 500 identifies the target database which holds the query dataset using the information held by the resource manager 600 in Figure 6. In this instance the dataset is identified as being held in the local dataset, and the query manager 500 constructs the query request and passes it to the query engine 510 for processing.

The query engine 510 issues in parallel a query against every instance of the local dataset. Each query generates a response which is passed back to the query manager 500. The query manager aggregates the responses, which in turn updates the query cache 515 and the query cache log 505. The query manager 500 returns the result to the application.

The query cache log 505 is used by the query manager 500 to maintain a history of the queries performed. This information is collated over the operating cycle, and in the event of a system restart, the query manager 500 identifies, for example, the 'top 10' most performed queries and performs pre-fetches of this data into the query cache. It will be appreciated by a person skilled in the art that the query manager could identifier any numbers of the most performed queries as configured by a system administrator.

A master broker 102, 106 may optionally broadcast and collate the most performed queries from all the participating database brokers 102, 106 such that the entire system may be pre-loaded with the most requested data at a restart of any database broker 102, 106.

Turning now to Figure 6, the monitoring component 215 of Figure 2 is illustrated. The monitoring component 215 receives requests from the configuration component 205 and provides the monitoring for the availability of one or more database brokers 102, 106. The monitoring component 215 comprises a resource manager 600, a master controller 605, broker probe

interface 610 and a monitoring interface 615. Each of these components will be explained in turn.

The resource manager 600 provides central control and management of the database broker 102, 106 resources such as which database brokers 102, 106 are active, what data sets are held where, and which database broker 102, 206 is a delegate for other database brokers 102, 106. The resource manager 600 sends the resource information to the master controller 605. The master controller 605 provides a central point of control and management for the monitoring of one or more database brokers 102, 106, and the handling of one or more remote database broker 102, 106 failures. If a database broker 102, 106 is acting as a master, the master controller provides the central point to which all other database brokers 102, 106 publish their availability. In the event that a master database broker 102, 106 fails then a secondary delegated master will take over this processing. The master controller 605 manages and interacts with two interfaces; the broker probe interface 610 handles the probing of one or more remote database brokers 102, 106, and the monitoring interface 615 which receives probes from other remote database brokers 102, 106. The master controller 615 further updates the resource manager 600 based on events received from one or more remote database brokers 102, 106 for example, when a database broker 102, 106 goes on or offline.

Lastly, the broker probe interface 610 provides the message management (send and receive functions) for the probing of one or more remote database brokers 102, 106 for example sending a message to the master controller informing that a new database broker has joined the system.

The database broker 102, 106 may be implemented as one of the following: an add on software module installed on an application server, connected via an API interface to an application layer or as a standalone broker service, connected via an API interface, an MQ series interface or other remote access mechanisms etc. MQ series is a registered trade mark of IBM Corporation in the U.S. and other countries.

The database broker 102, 106 may operate in two modes: delegated mode and master mode. Delegated mode allows the data to be distributed across multiple geographical sites. Each database broker 102, 106 will hold a copy of every database required to provide a web service and each database broker 102, 106 within a particular geographical location is given delegated authority for owning a portion of the data (a dataset) in the update/retrieval mechanism 220. In order to allow a delegate broker 102, 106 to update the data within

the database, a database schema is devised such that the database may be spilt into common core data and a number of data stores (local data) which are relevant to the server providing the web service. This is illustrated in Figure 7, wherein an update/retrieval mechanism comprises a plurality of common databases 700, 705, a configuration database and a plurality of local databases 715 and 720.

An example of this type of database schema can be explained using an example of an on-line shop. An on-line shop will sell a variety of goods. The on-line shop may have a customer base in the United Kingdom, the United States and Italy. In this example the common data will comprise the variety of goods that the on-line store sells for example door handles, sinks and taps, plus other information such as the quantity of these items in stock and the price of each of these items. In the delegate model only the master database broker will be have the authority to update the common data for the UK, United States and Italy.

The local data 715, 720 comprises data that is relevant to the server in a geographical region. Using the example of the on-line shop the local data may be the account details of the customer's i.e. names and delivery addresses for the UK, US and Italy. The UK database broker 102 is the owner of the dataset of the UK customer account details, the US database broker 106 is the owner of the dataset for the US customer account details and the Italian database broker is the owner of the dataset for the Italian customer account details.

In another embodiment the database broker 102, 106 may also work in 'master mode'. This mode requires all database updates to be authorized centrally by the master database broker 102, 106. If any of the websites are required to update a record within a dataset the master database broker 102, 106 will be the sole arbiter for all update requests. If at any time the master database broker 102, 106 becomes inactive a designated secondary database broker will assume the role of the master database broker and will authorize update requests.

The designation of a secondary delegate master database broker is explained in greater detail with reference to Figure 11.

Turning to Figure 8, Figure 8 depicts the segmentation of data at multiple geographical locations 10, 12 and 13. Communication between the database brokers 102, 106 is carried out over a private network 108. For each

web service there may be common data for example, the availability of stock. The common data is distributed across three databases at site 1 10, site 2 12 and site 3 13. A master database broker 800 claims ownership of the common database 810 and it is only the master database broker 800 that may perform updates on the common database 810. Site 2 825 and site 3 830 own instances 820 and 825 (identical copies of the database) and the database brokers 815, 830 of site 2 12 and site 3 13 are required to package their updates and send to the master database broker 800 requesting an update to be performed. Each packaged update is given a sequence number and an update will be performed in an order dictated by the sequence number.

Referring to Figure 9, the steps that a database broker performs when carrying out an update are shown. The database broker receives a request from the application interface at step 900. The database update is extracted from request and at step 905 it is determined whether the database update is intended for the common database or a local database. If the database update is for the common database, control passes to step 910 where it is determined whether the database broker dealing with the current request is the master database broker or a secondary database broker. If at step 910 it is determined that the database broker is the master database broker for the common database, control passes to step 915 and the master database broker updates the relevant record in the master common database. To ensure that all database brokers are able to view the update, a database update message is sent to all database brokers informing them of the update and a confirmation receipt is sent back to the master broker.

Returning back to step 905, if the determining step determines that the database update is for a local database, control passes to step 930 and the owner is identified for the dataset that is required to be updated. If the database broker receiving the request at step 900 is the owner of the dataset the update can be performed on the dataset within the local database.

If at step 930 the database broker receiving the request at step 900 is not the owner of the dataset control passes to step 935 and a lookup may be performed to determine who the owner of the dataset is and an update message is sent to the owning database broker at step 940 requesting the dataset to be updated at step 925.

Returning to step 910, if it is determined that the database broker receiving the update request at step 900 is not the master broker but a secondary broker control passes to step 920 and a request message is sent to

the master of the common database requesting that the dataset is updated at step 925. A sequence number is created by the sequence manager and an update is performed to the dataset at step 925.

Referring to Figure 10 when the server is initiated 1 the master database broker registers itself at step 2 as the master database broker in the configuration database and obtains a list of registered secondary database brokers at step 3. As other secondary database brokers can be added to the system at any time, a check is continually performed to determine if any secondary database brokers have been added. Once it has been determined that a secondary broker has been added, an active broker list and a database status list is updated with the new information.

At step 4 a probe is initiated to determine whether all the registered database brokers are active. The active list of database brokers is loaded into the system to enable the probe to monitor all active registered secondary database brokers.

The probe sends a message every n seconds to each registered database broker probe in the active list to determine at step 6 that it is still active and able to process database updates. A probe status list is created to store the status of each probe for example whether a response has been returned to the probe.

The monitoring component receives the probe status list from the probe and determines if a response has been received from each secondary broker listed in the active list. If a response has not been received from any one of the secondary database brokers, the monitoring component determines if a probe has reached its maximum allotted time at step 6 in which to respond for example 20 milliseconds. It will be appreciated by a person skilled in the art that the maximum allotted time is not limited to 20 seconds but can be of a lesser or greater value.

If the secondary database broker has not reached its maximum allotted time control passes to step 5 and the probe is re-issued. If it has reached its allotted time and no response has been received, control passes to step 9 and the secondary database broker is removed from the active list and the status is updated in the database status list.

If the probe does not receive a response a further check is carried out at step 7 to determine if it is the master database broker or a secondary

database broker that is failing. Once it has been determined that it is the master database broker that is failing, a new master delegate is located from the delegate configuration database at step 8 and the new master delegate broker is located and it registers itself with the configuration database and all other active database brokers are informed of the update.

The database broker architecture supports resilience and failover when one of the web sites or web services is lost through the implementation of a chain of command i.e. if the master database broker is lost then the master database broker hands down authority to update the common database to a designated secondary broker which then takes over the role as the master database broker.

A chain of command as stored in the configuration database comprises a plurality of hierarchical relationships which many be created between different database brokers 102, 106 for example with reference to figure 11, web site XYZ.co.uk 111, uses four web servers: server 1 112, server 2 113, server 3 114 and server 4 115. In this example, the database broker of server 1 112 is registered as the master server. The database brokers of server 2 113, server 3 114 and server 4 115 are all registered in the configuration database as delegate brokers and loaded into the active list as active delegate brokers. The active list further defines the relationships between the delegate brokers for example if server 1 fails, it is possible to define that server 2 113, will take over the role of the master server from server 1 112, if the probe establishes that server 1 112 is no longer active. Using the example in Figure 11, the same rules may be defined for server 2 113, server 3 114 and server 4 115. If it is found at step 7 of figure 10 that server 2 113 is failing, the database broker of server 3 114 can register itself as the master broker and takes over from server 2 113, the same applies to the database brokers of server 3 114 and server 4 115.

More complicated relationships can be defined, for example, with reference to the website XYZ.com 118 of Figure 11. Server 119 is registered as the master broker, if the probe finds that server 5 119 is failing, control passes to the database broker of server 7 121 or server 6 120 depending which one is available. If the database broker of server 6 120 assumes responsibility as the master database broker, server 7 121 or server 8 123 may take over depending on availability.

It will be appreciated by a person skilled in the art that any number of relationships can be created not just within one website in one geographical

location but relationships across multiple geographic locations for example US and UK locations.

5

CLAIMS

1. A method for the synchronisation of data in a geographically load-balanced environment comprising a database broker, the database broker being operable as a middleware service, the method comprising the steps of:

receiving a request from an application layer to update a dataset owned by the database broker, the dataset being for use by an update mechanism;

first determining if the dataset for use by the update mechanism belongs to a common database or a local database;

second determining if the database broker receiving the request is a master database broker or a secondary database broker; and

in response to a first and second successful determining step the owning database broker updating the dataset in the common database or the local database.

2. A method as claimed in claim 1 wherein the update mechanism comprises a stored instance of each database required to provide a service.

3. A method as claimed in claim 2 wherein the stored instance of each database is segmented by database broker ownership.

4. A method as claimed in claim 3 wherein a database broker may only update a dataset owned by itself.

5. A method as claimed in claim 1 wherein a master database broker is the owner of the common database.

6. A method as claimed in claim 1 wherein a secondary database broker is the owner of a dataset within a local database.

7. A method as claimed in claim 5 wherein the master database broker has the authority to update a dataset stored within the common database in the update/retrieval mechanism.

8. A method as claimed in claim 1 comprises the further step of probing the database broker to determine if the database broker is active.

9. A method as claimed in claim 1 wherein the database broker registers itself as a master database broker or a secondary database broker and registers its status as active.

10. A method as claimed in claim 8 wherein the probe sends a message to a registered database broker to determine if the registered database broker is still active.

11. A method as claimed in claim 10 wherein in response to an unsuccessful determining step the registered database broker is removed from a list of registered active database brokers.

12. A method as claimed in claim 11 wherein the probe further determines in response to an unsuccessful determining step, if the registered database broker is the registered master database broker.

13. A method as claimed in claim 12 wherein in response to a successful determining step identifying a delegate database broker to replace the registered master database broker.

14. A method as claimed in claim 1 wherein the database broker maintains a log of the most performed queries held in its cache.

15. A method as claimed in claim 14, wherein the log identifies the at least one most common query performed.

16. A method as claimed in claim 15 wherein the database broker pre-loads the at least one identified most common query into the cache.

17. A method as claimed in claim 9 or claim 15 wherein the master database broker aggregates the at least one most common query from all registered database brokers.

18. A method as claimed in claim 17 wherein the master database broker propagates the aggregation of the at least one most common query to all registered database brokers.

19. A method as claimed in claim 1 wherein the update mechanism comprises a retrieval mechanism for retrieving a dataset in response to a query.

20. A database broker system for the synchronization of data in a geographically load-balanced environment, the database broker system being operable as a middleware service, the system comprising:

a query module comprising means for managing queries received from an application layer and sending and receiving information from an update mechanism;

an update module comprising means for synchronizing at least one requested update between the at least one database broker;

a configuration module comprising means for receiving data from an application layer and storing configuration data about the database broker and means for receiving updates from a monitoring component; and

a monitoring module comprising means for determining if the database broker is active and means for sending one or more updates to the update module.

20. A system as claimed in claim 19 further comprises means for interfacing with an update and/or retrieval mechanism.

21. A system as claimed in claim 20 wherein the update or retrieval mechanism comprises means for the segmentation of at least one database into to logical datasets.

22. A system as claimed in claim 19 wherein the monitoring component further comprises means for probing the database broker and means for determining if the database broker is active.

23. A system as claimed in claim 22 wherein the monitoring component further comprises means for determining if the master database broker is failing.

24. A system as claimed in claim 23 wherein means for determining further comprises locating a delegate database broker in response to a successful determining means.

25. A system as claimed in claim 23 wherein means for locating a delegate database broker is determined by means for the configuration module defining one or more delegate relationships.

26. A system as claimed in claim 18 wherein the database broker provides means for maintaining a log of the most performed queries held in its cache.

27. A system as claimed in claim 26, wherein the log provides means for identifying the at least one most common query performed.

28. A system as claimed in claim 27 wherein the database broker provides means for pre-loading the at least one identified most common query into the cache.

29. A system as claimed in claim 19 or claim 27 wherein the master broker provides means for the aggregation of the at least one most common query from all registered database brokers.

30. A method as claimed in claim 29 wherein the master database broker propagates the aggregation of the at least one most common query to all database brokers.

31. A computer program product comprising computer program code stored on a computer readable storage medium, which when executed on a data processing system, instructs the data processing system to carry out the method as claimed in claim 1.

32. A data synchronisation service in a geographically load balanced environment comprising a database broker, the database broker being implemented as a middleware service, the service comprising the steps of:

receiving a request from an application layer to update a dataset owned by the database broker, the dataset being for use by an update mechanism;

first determining if the dataset for use by the update mechanism belongs to a common database or a local database;

second determining if the database broker receiving the request is a master database broker or a secondary database broker; and

in response to a first and second successful determining step the owning database broker updating the dataset in the common database or the local database.

ABSTRACT

A DATA BROKERING METHOD AND SYSTEM

A method for the synchronisation of data in a geographically load-balanced environment comprising a database broker, the database broker being operable as a middleware service, the method comprising the steps of: receiving a request from an application layer to update a dataset owned by the database broker, the dataset being for use by an update mechanism; first determining if the dataset for use by the update mechanism belongs to a common database or a local database; second determining if the database broker receiving the request is a master database broker or a secondary database broker; and in response to a first and second successful determining step the owning database broker updating the dataset in the common database or the local database.

Fig. 1

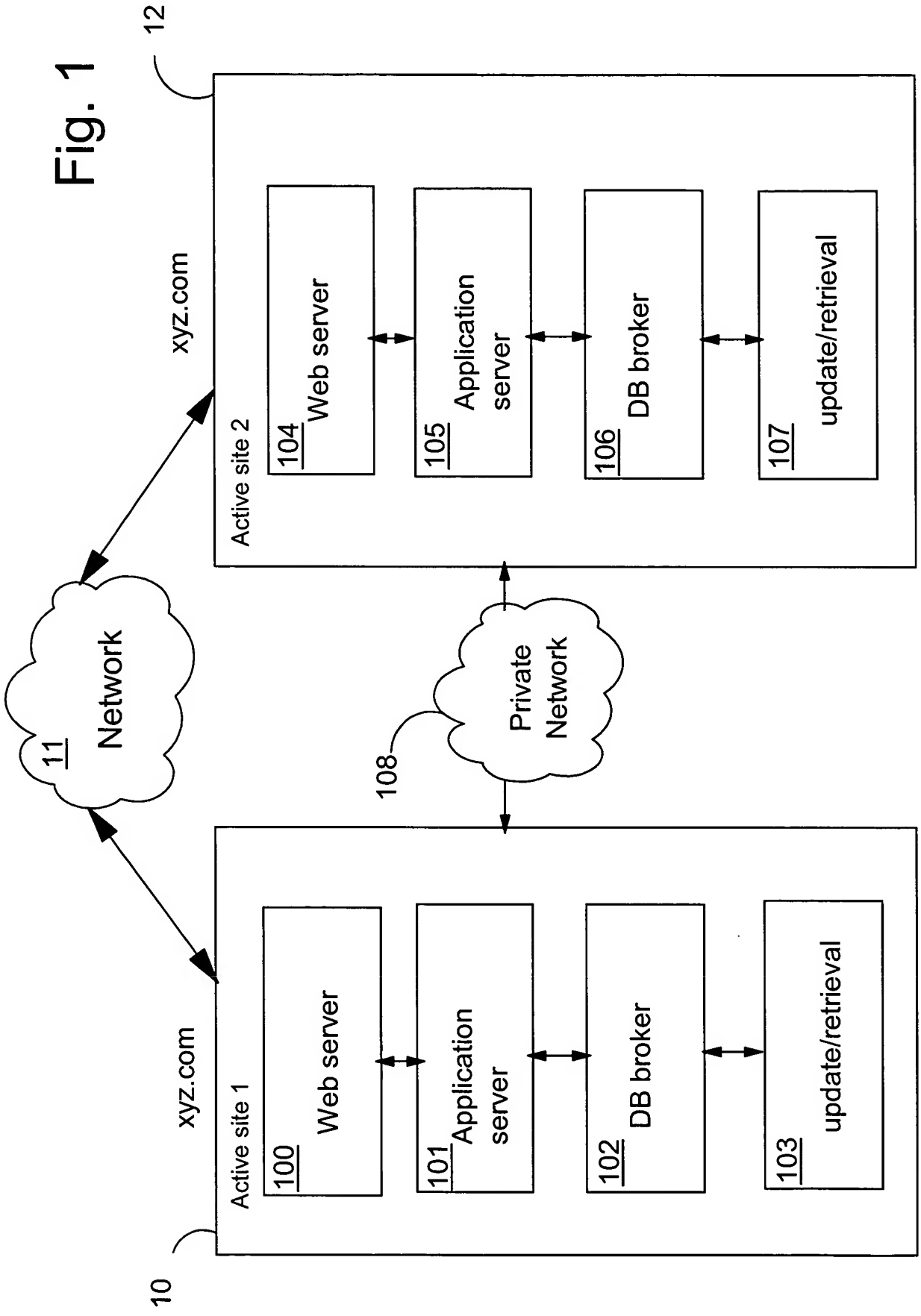

10
12
xyz.com
Active site 1
100 Web server
101 Application server
102 DB broker
103 update/retrieval
xyz.com
Active site 2
104 Web server
105 Application server
106 DB broker
107 update/retrieval
11 Network
108 Private Network

Fig. 2

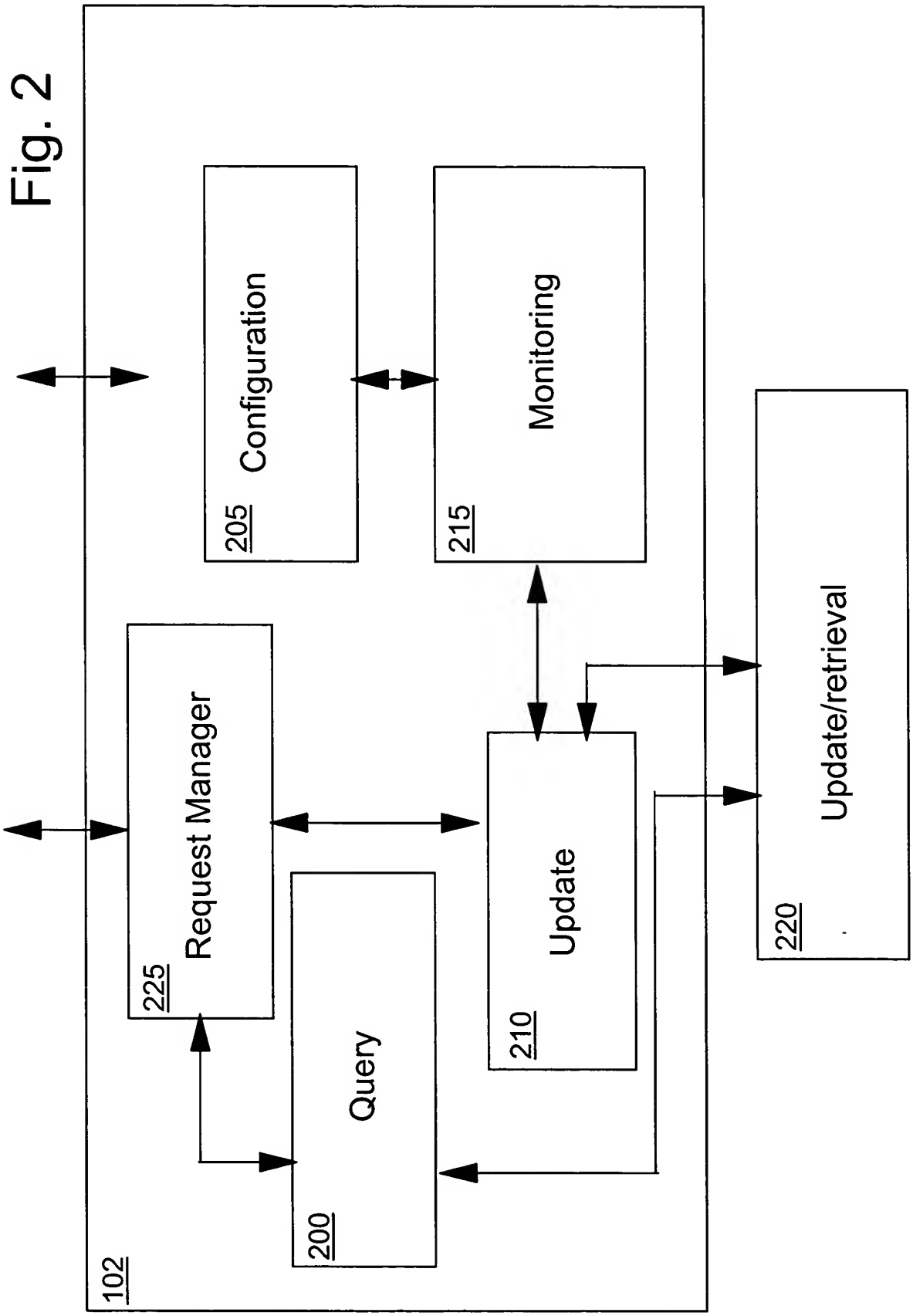

102
205
Configuration
215
Monitoring
225
Request Manager
210
Update
220
Update/retrieval
200
Query

Fig. 3

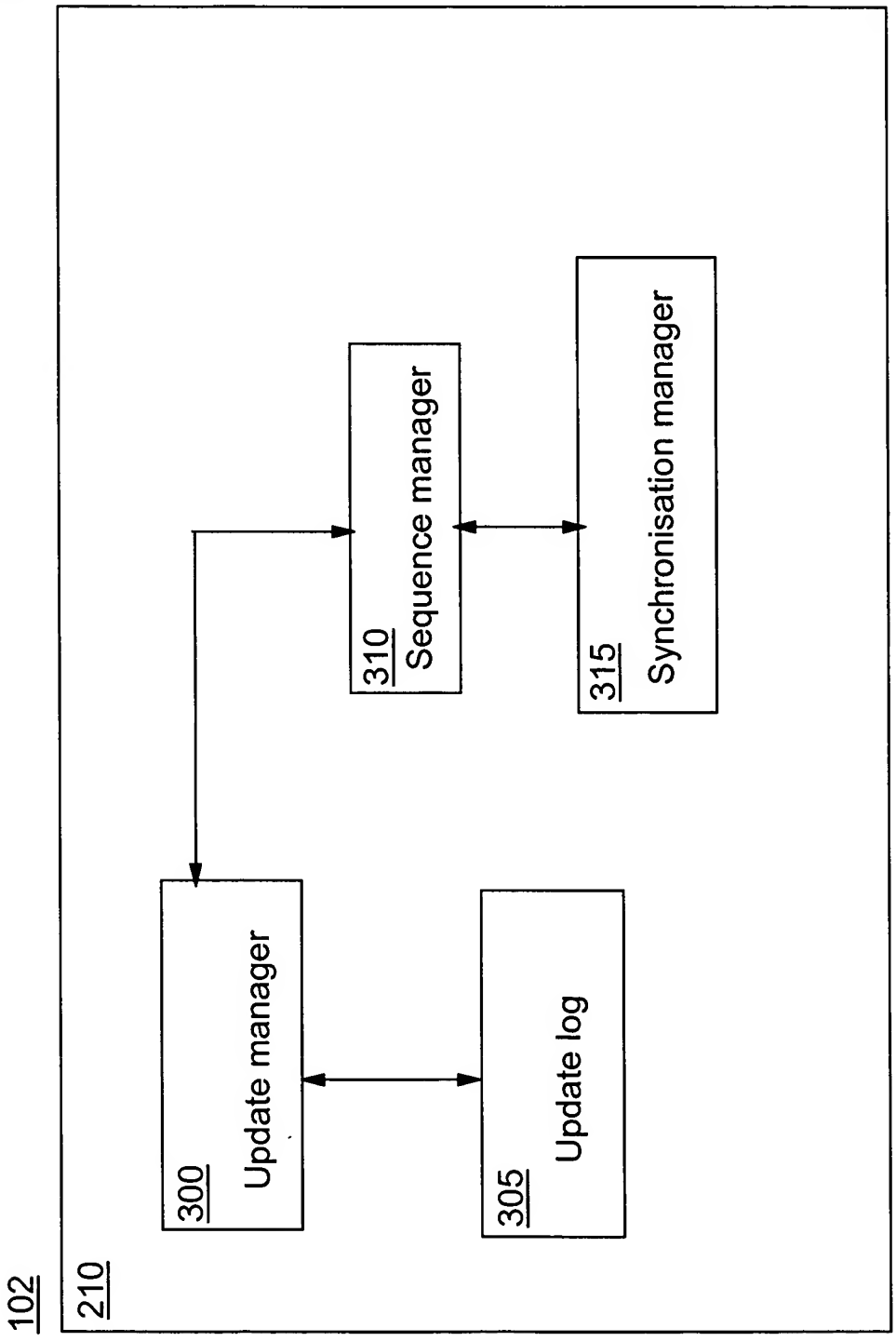

102
210
300
Update manager
305
Update log
310
Sequence manager
315
Synchronisation manager

Fig. 4

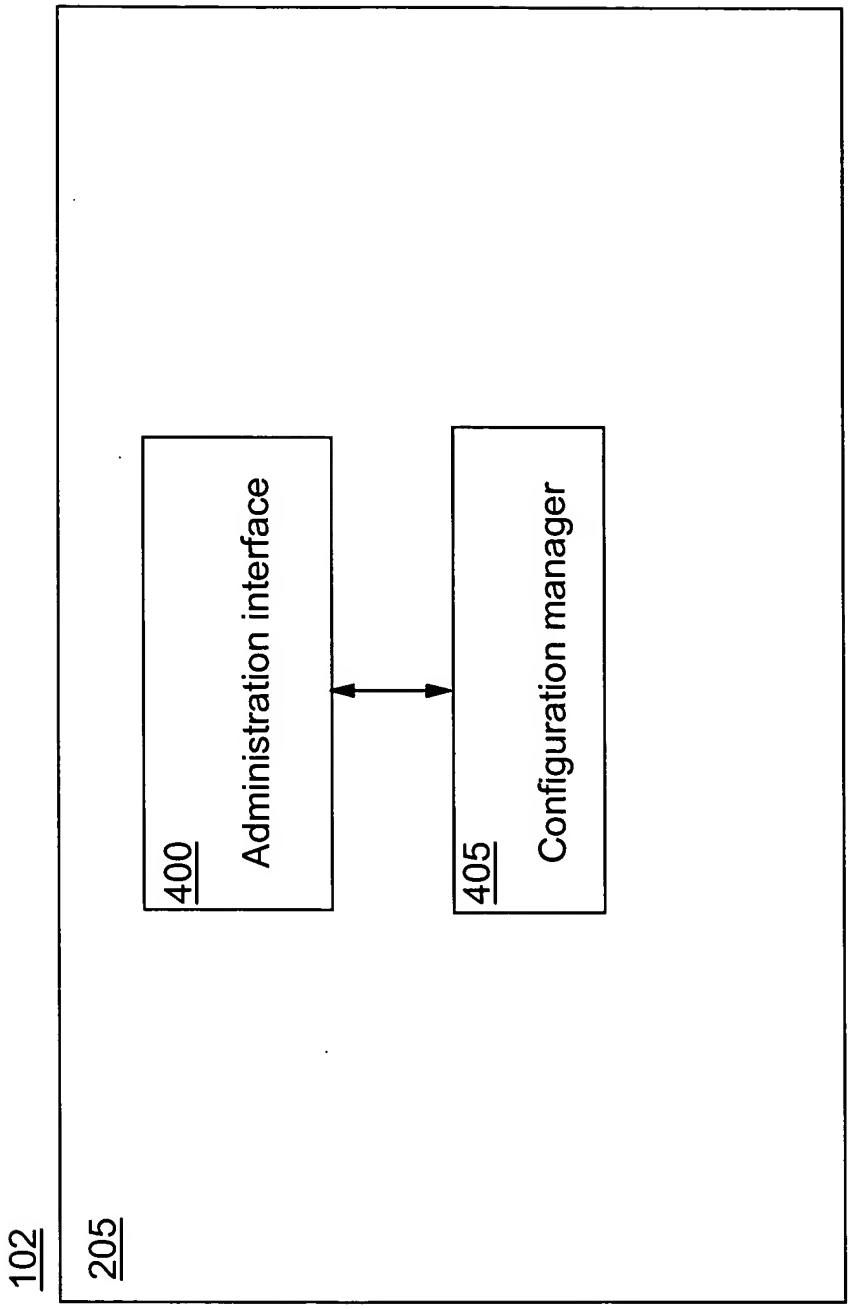

102
205
400
Administration interface
405
Configuration manager

Fig. 5

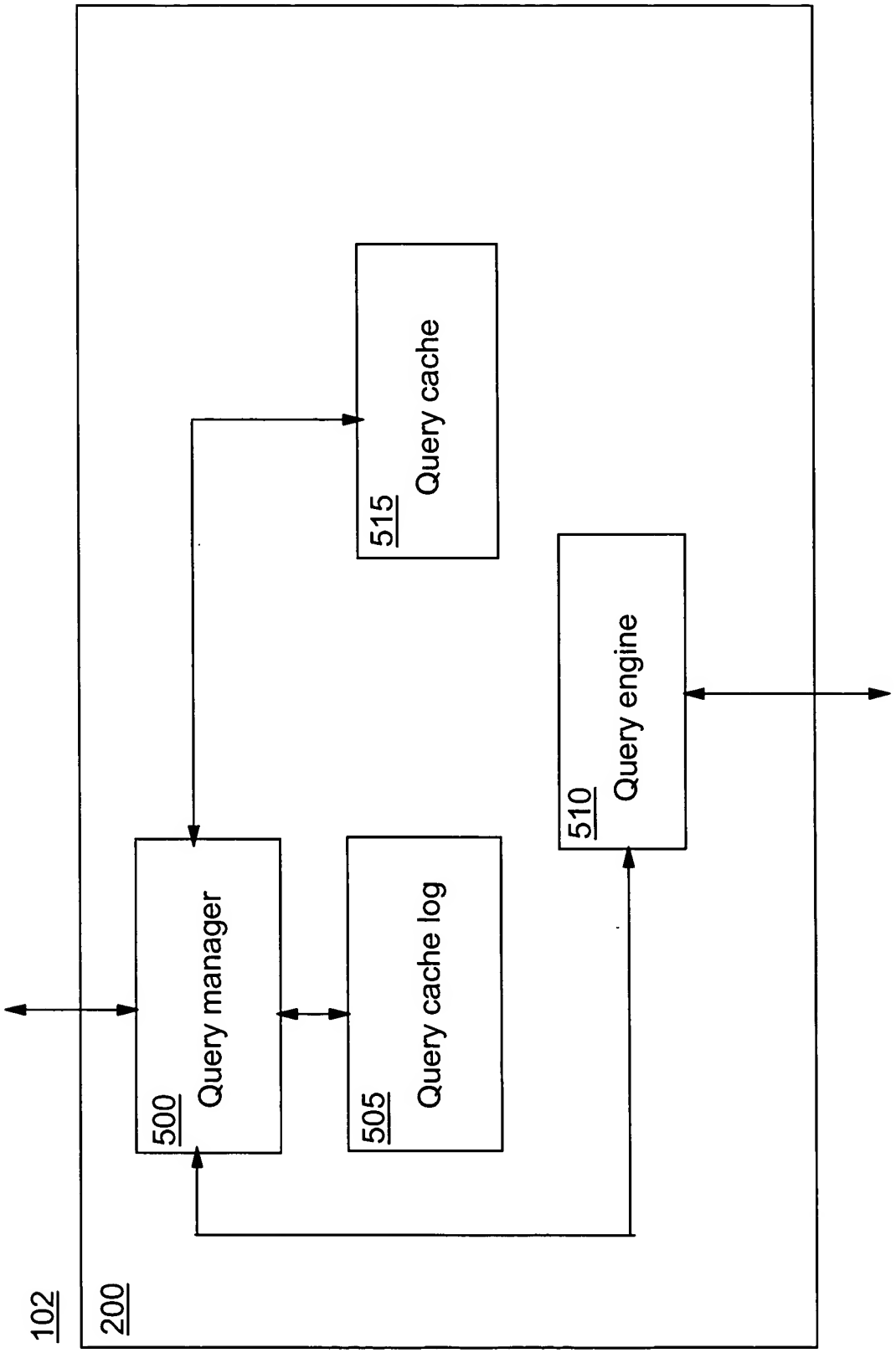

102
200
500
Query manager
505
Query cache log
515
Query cache
510
Query engine

Fig. 6

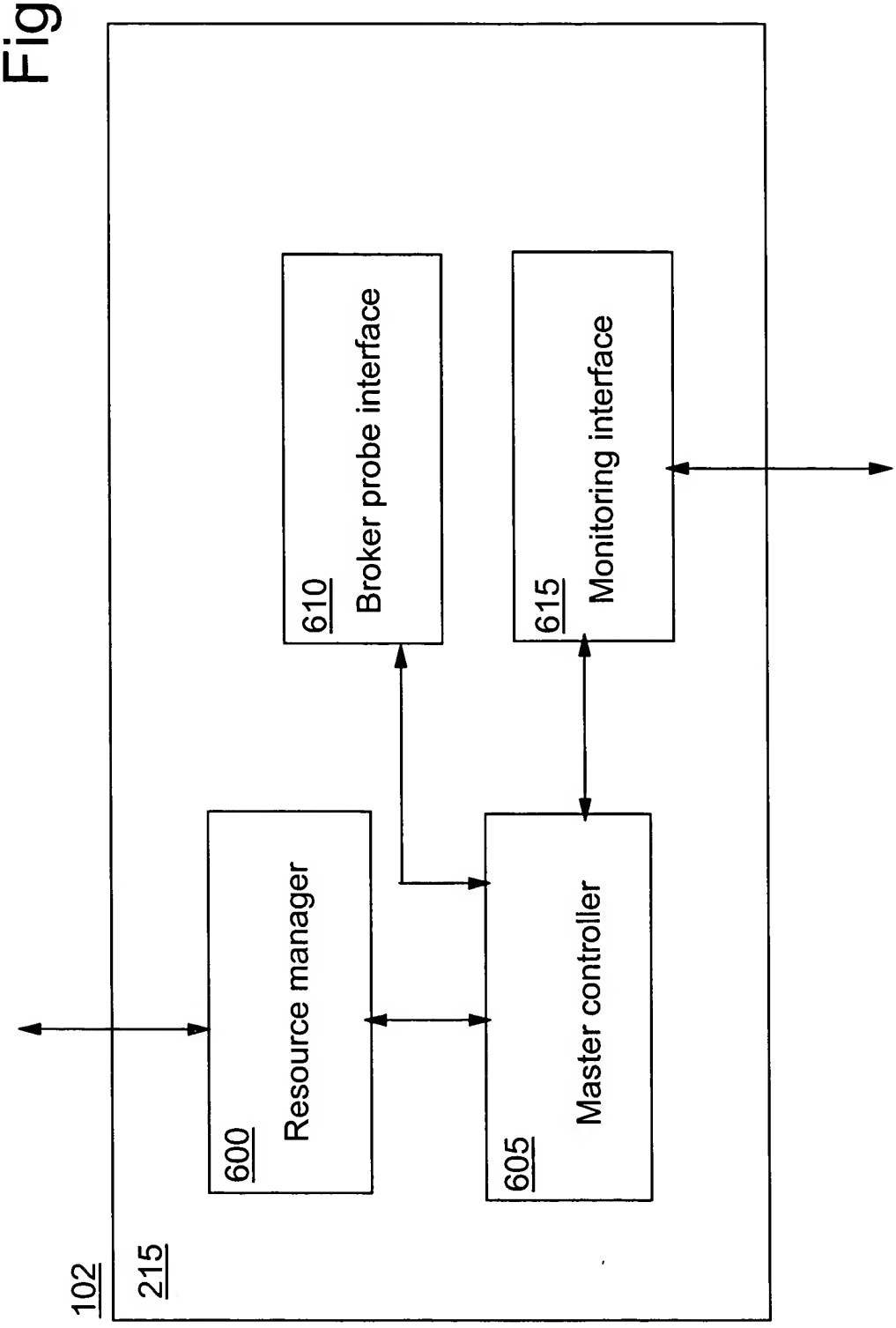

102
215
600
Resource manager
605
Master controller
610
Broker probe interface
615
Monitoring interface

Fig. 7

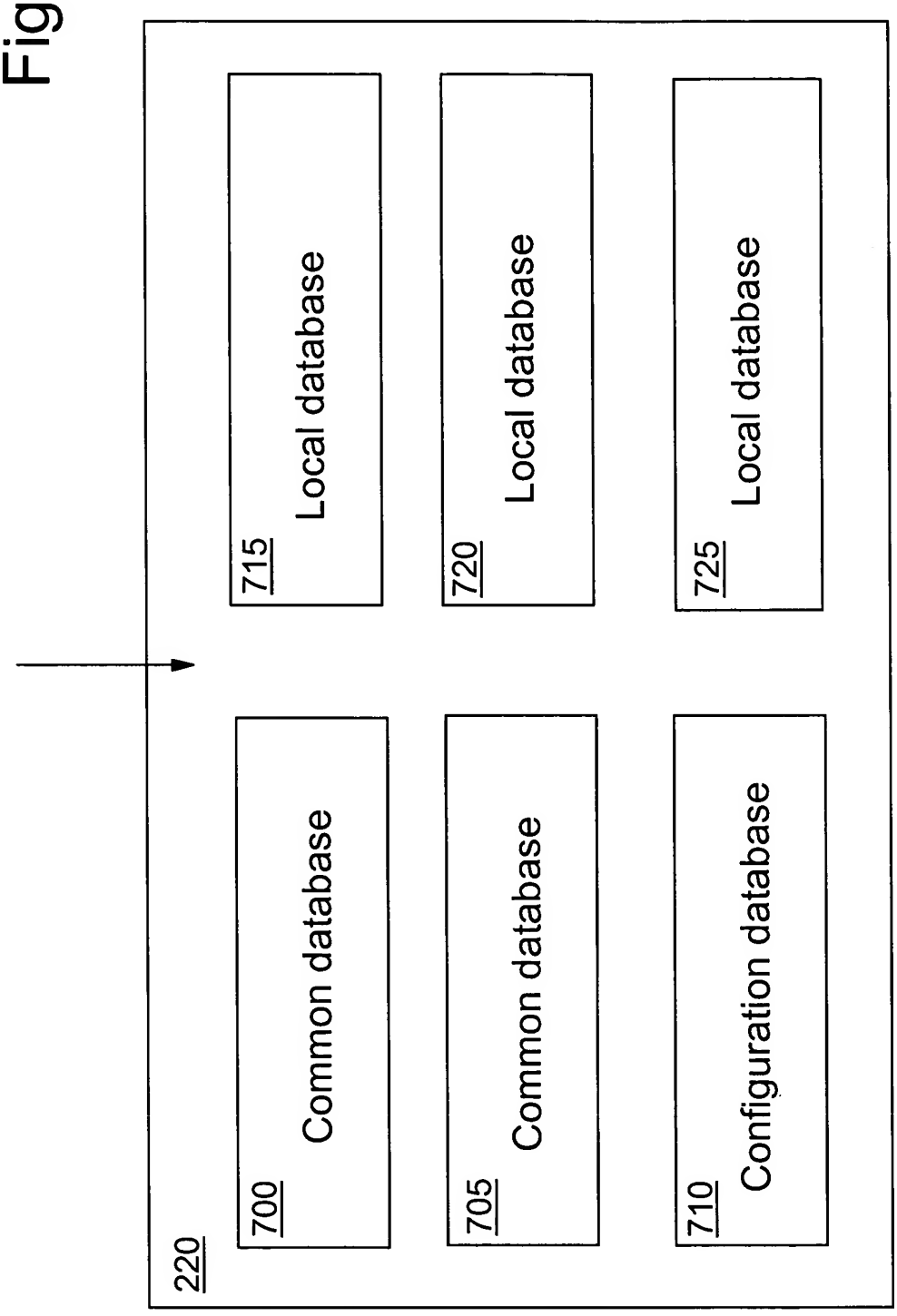

220
700
Common database
705
Common database
710
Configuration database
715
Local database
720
Local database
725
Local database

Fig. 8

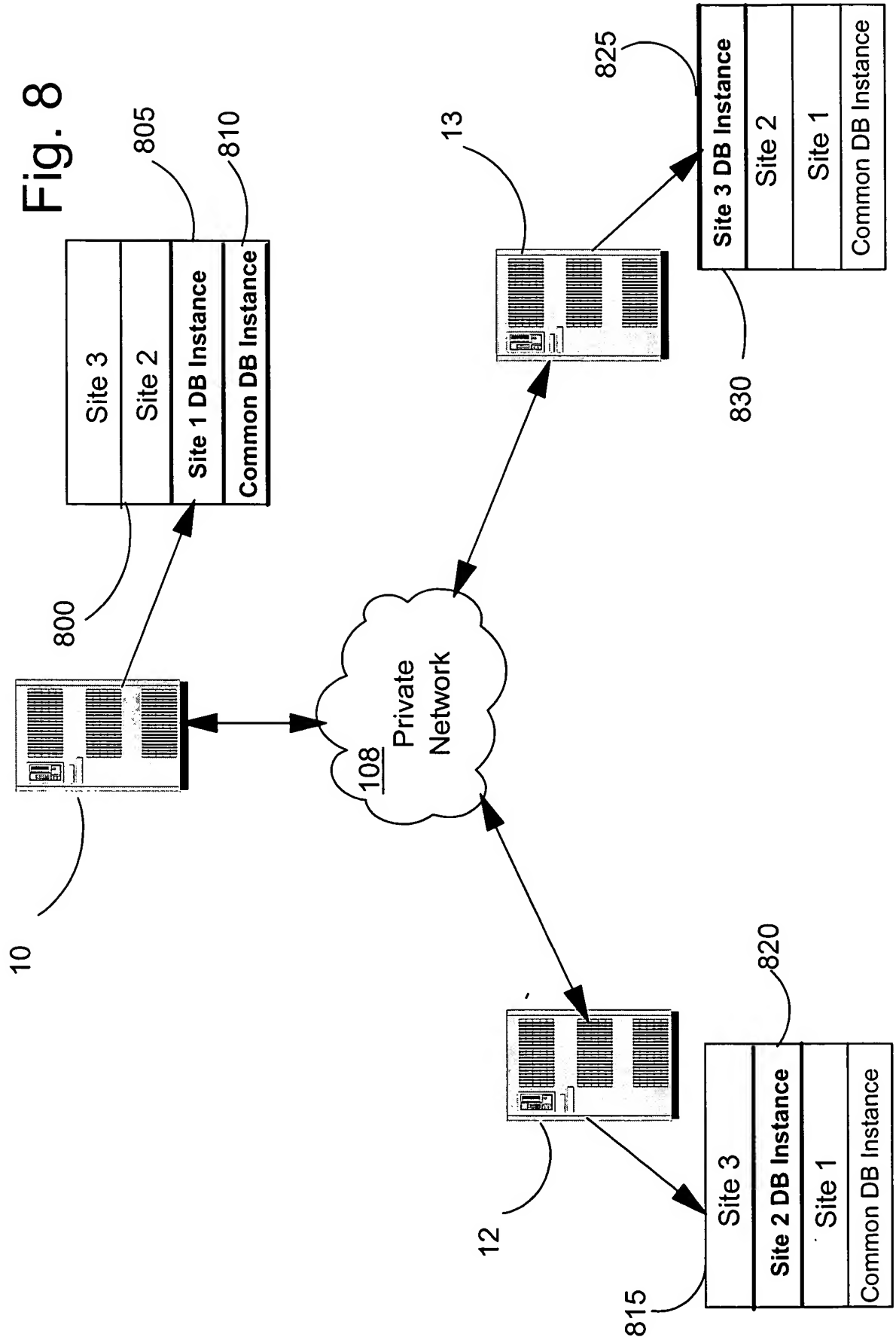

Site 3
Site 2
Site 1 DB Instance
Common DB Instance
805
810
800
10
Private Network
108
13
825
Site 3 DB Instance
Site 2
Site 1
Common DB Instance
830
12
820
Site 3
Site 2 DB Instance
Site 1
Common DB Instance
815

Fig. 9

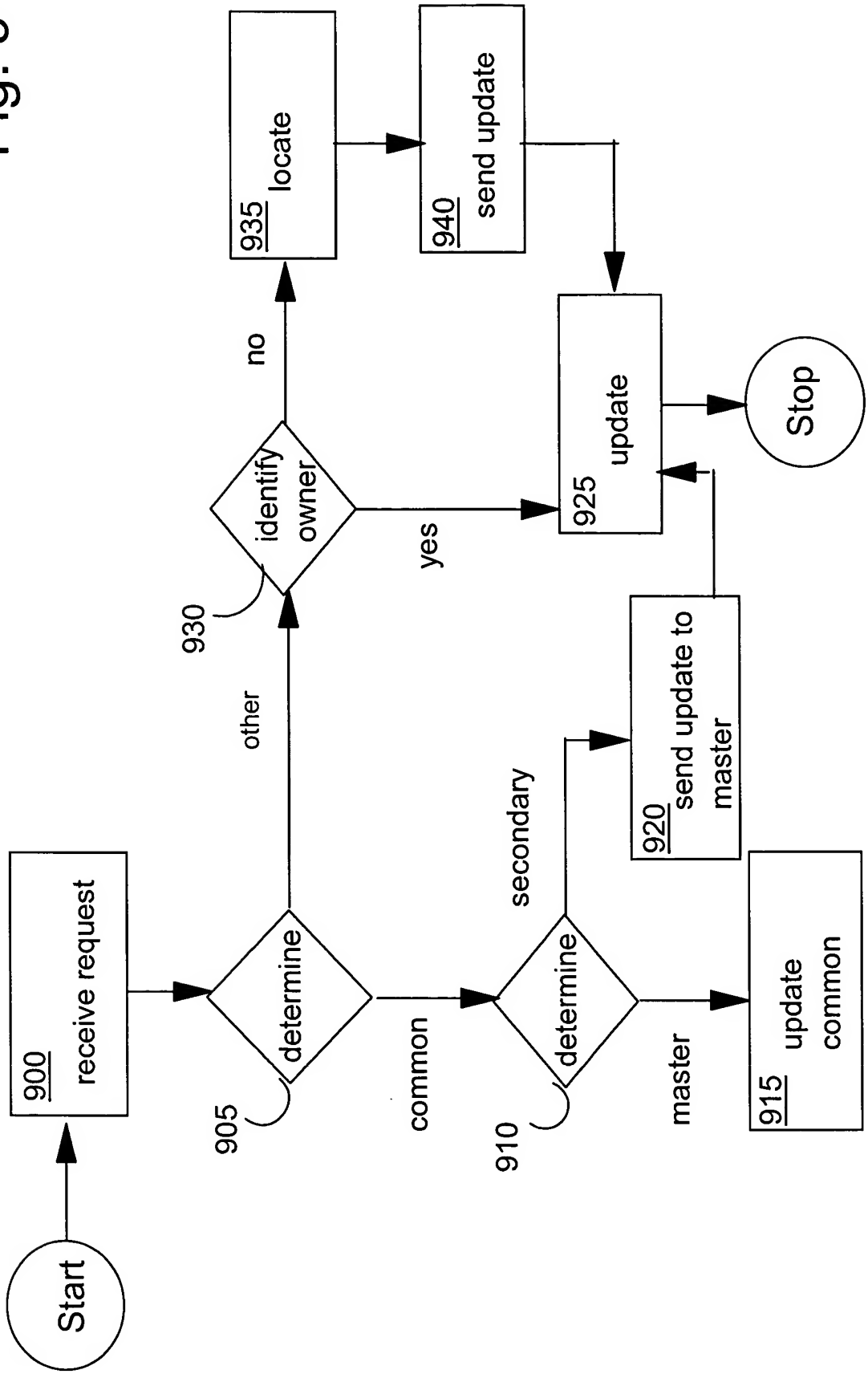

Start
900 receive request
905 determine
other
common
930 identify owner
no
935 locate
940 send update
yes
910 determine
secondary
920 send update to master
master
915 update common
925 update
Stop

Fig. 10

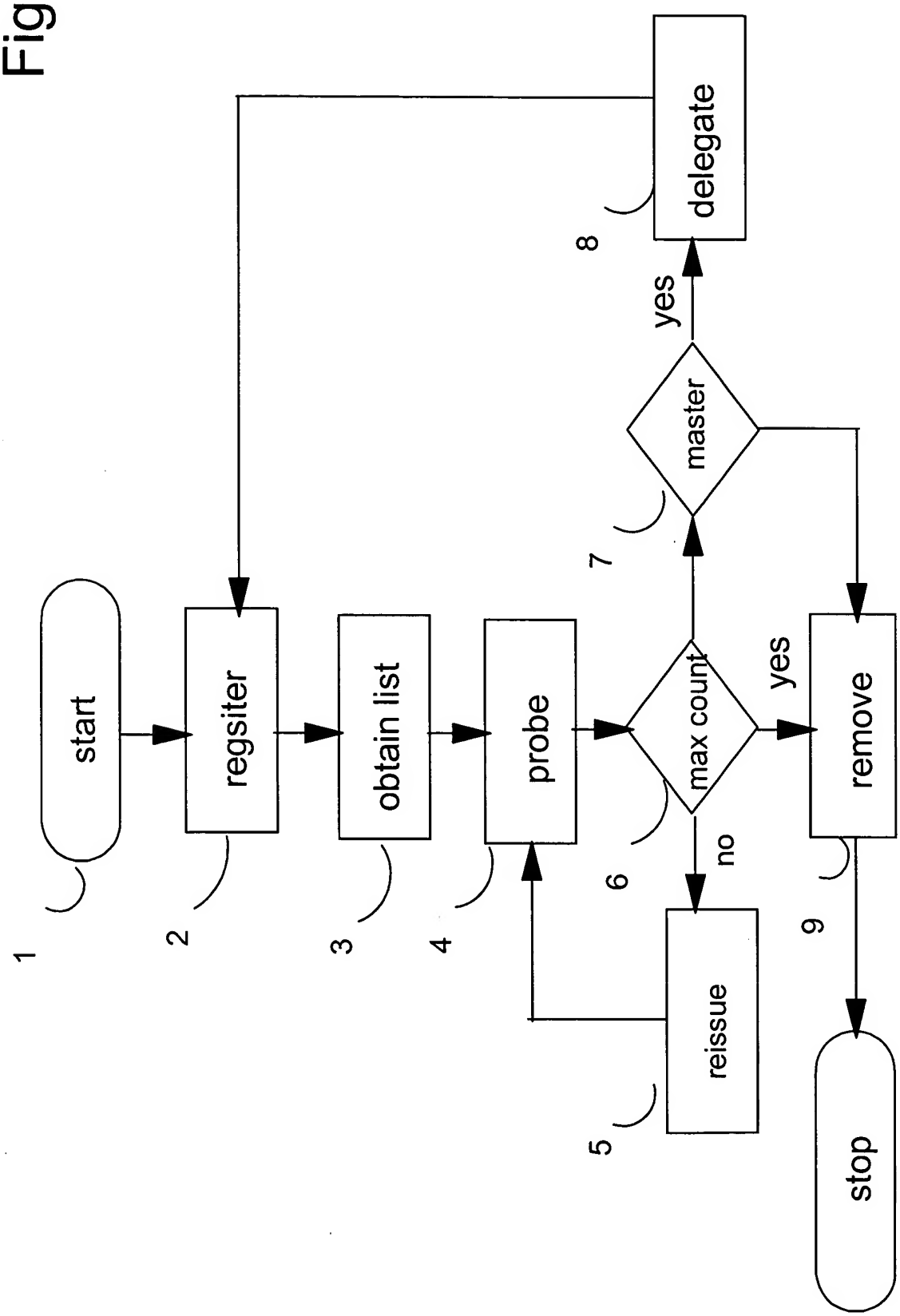
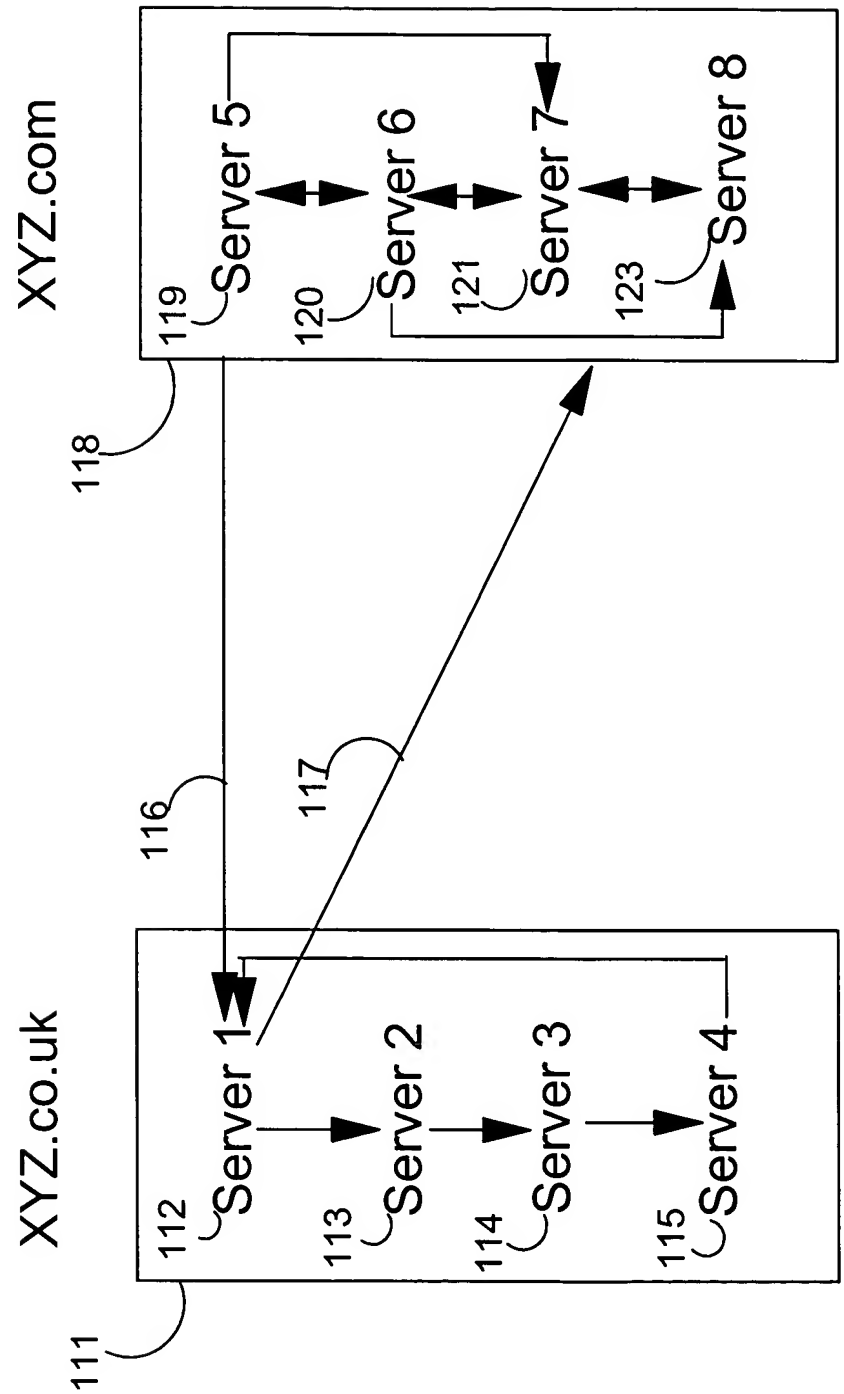

start
regsiter
obtain list
probe
max count
yes
remove
stop
no
reissue
master
yes
delegate
1
2
3
4
5
6
7
8
9

Fig. 11


XYZ.com
XYZ.co.uk
Server 1
Server 2
Server 3
Server 4
Server 5
Server 6
Server 7
Server 8
111
112
113
114
115
116
117
118
119
120
121
123